Orchestrating Bulk Data Movement in Grid Environments

Sudharshan Vazhkudai
vazhkudaiss@ornl.gov
Computer Science and Mathematics Division
Oak Ridge National Laboratory

Abstract

Data Grids provide a convenient environment for researchers to manage and access massively distributed bulk data by addressing several system and transfer challenges inherent to these environments. This work addresses issues involved in the efficient selection and access of replicated data in Grid environments in the context of the Globus Toolkit™, building middleware that (1) selects datasets in highly replicated environments, enabling efficient scheduling of data transfer requests; (2) predicts transfer times of bulk wide-area data transfers using extensive statistical analysis; and (3) co-allocates bulk data transfer requests, enabling parallel downloads from mirrored sites. These efforts have demonstrated a decentralized data scheduling architecture, a set of forecasting tools that predict bandwidth availability within 15% error and co-allocation architecture, and heuristics that expedites data downloads by up to 2 times.

Keywords: *Data Grids, Replica Selection, Predictions, Co-Allocations, Scheduling, Parallel Downloading, Data Movement, Middleware.*

1.0. Introduction

As the coordinated use of distributed resources, or Grid computing, becomes more commonplace, basic resource usage is changing. Many recent applications use Grid systems as distributed data stores [DataGrid02, GriPhyN02, HSS00, LIGO02, MMR+01, NM02], where pieces of large datasets are replicated over several sites. For example, several high-energy physics experiments have agreed on a tiered Data Grid architecture [HJS+00, Holtman00] in which all data (approximately 20 petabytes by 2006) is located at a single Tier 0 site; various (overlapping) subsets of this data are located at national Tier 1 sites, each with roughly one-tenth the capacity; smaller subsets are cached at smaller Tier 2 regional sites; and so on. Therefore, any particular dataset is likely to have replicas located at multiple sites [RF01, LSZ+02, LSZ+03]. Such replicated, bulk data is required to be moved across the Grid in response to user queries.

Moving bulk data in Grid environments is mired by several nontrivial issues. First, information on the state and behavior of replica/storage locations needs to be identified, gathered and exposed so they can be discovered in the context of an information service. Second, suitable locations from which to fetch the dataset needs to be identified based on heuristics that combine user specified requirements/policies and current system behavioral trends. Further, the scheduling needs to be accomplished in a decentralized fashion so it can scale to Grid-proportions. Third, quality of service agreements, negotiations and guarantees—pin replicas for future access, reserve storage to stage data in the future, or reserve network bandwidth—need to be coordinated to maximize usage. Fourth, auto-tuning of data movement protocols—setting TCP buffers or parallel streams—needs to be enabled so data transfer throughput can be maximized. Large data transfers with default operating system buffer sizes (64 KB) and no parallelism will almost certainly result in poor rates rendering them unsuitable for Grid environments. Finally, the bulk data transfer should be monitored for progress and deterioration so corrective measures can be taken in case of performance degradation. Thus, we need performance metrics to determine transfer degradation.

Aforementioned are but a few—yet critical—set of activities in Grid scheduling. Bulk data movement is usually associated with some computation to be run alongside, leading to a larger question of finding schedules for both data and computation. In this paper, however, we address a few steps essential to orchestrate the movement of massively replicated bulk data (of the order of several hundred megabytes to gigabytes) in Grid environments, bringing together, in one fold, several of our previous work. We present our work in the context of the Globus Toolkit™

[FK98, Globus02]. The Globus Data Grid provides a convenient architecture for scientific experimental groups to share and democratize data access.

We discuss the design and implementation of a scalable, decentralized replica selection service that uses information regarding replica location and user preferences to guide selection from among storage replica alternatives [VT00, VTF01]. In Section 2 we first present a basic replica selection service design, then show how dynamic information collected using Globus information service capabilities concerning storage system properties can help improve and optimize the selection process. We demonstrate the use of Condor's [LLM88] ClassAds [RLS98] resource description and matchmaking mechanism as an efficient tool for representing and matching storage resource capabilities and policies against application requirements.

Different sites may have varying performance characteristics because of diverse storage system architectures, network connectivity features, or load characteristics. Users (or brokers acting on their behalf) may want to be able to determine the site from which particular data sets can be retrieved most efficiently, especially as data sets of interest tend to be large (1–1000 MB). Since large file transfers can be costly, there is a significant benefit in selecting the most appropriate replica for a given set of constraints [ACF+02, VTF01]. One way a more intelligent replica selection can be achieved is by having replica locations expose performance information about past data transfers. This information can, in theory, provide a reasonable approximation of the end-to-end throughput for a particular transfer. It can then be used to make predictions about the future behavior between the sites involved. In our work we use GridFTP [AFN+01], part of the Globus Toolkit^{TM} for moving data, and present predictions using regression techniques to forecast the performance of GridFTP transfers for large files across the Grid [VS03]. Our approach combines end-to-end application throughput observations with network and disk load variations and captures whole-system performance and fluctuations in load patterns. Our predictions characterize the effect of load variations of several shared devices (network and disk) on file transfer times. In Section 3 we develop a suite of univariate and multivariate predictors that can use multiple data sources to improve the accuracy of the predictions as well as address Data Grid variations (availability of data and sporadic nature of transfers). These predictions are thematic to the replica selection problem. Our results indicate a prediction accuracy of 15% error for the testbed sites.

Data locations, once identified and ranked using our selection and prediction techniques respectively, can then be grouped together to collectively deliver the replica. Due to varied performance characteristics of the several replica locations and the transient load in the links connecting them to the client, downloading datasets even from the best of servers can often result in pedestrian transfer rates. A promising alternative is to download data from multiple locations, establishing multiple connections in parallel. With this approach, instead of downloading the entire dataset from a single sever, unique partial copies of the dataset are fetched from multiple servers in parallel that are later reassembled at the client end. In Section 4 we develop a basic architecture for co-allocating Grid data transfers and build a few techniques for downloading data in parallel, from multiple servers [Vazhkudai03]. We develop three techniques: (1) brute force co-allocation, (2) history-based co-allocation of flows and (3) dynamic load balancing. We apply these techniques to the GridFTP data movement tool, and evaluate our approaches by conducting performance analysis experiments in a wide-area testbed. Our results indicate a significant increase in bandwidth due to distributed downloads and denote that dynamic solutions outperform static approaches.

2.0. Scheduling Grid Data Transfers—Replica Selection

2.1. Scheduling Architecture

Traditionally, resource brokers have adopted a *centralized* approach to resource management, wherein a single node is responsible for decision making. An example of such an environment is the Condor [LLM88] high-throughput computing platform, wherein a central manager is responsible for matching resources against jobs. Obvious disadvantages to this approach are scalability and a single point of failure. Of course, Condor has an efficient recovery mechanism to address failure and has been proven to scale to thousands of resources and users.

But there is a more fundamental problem with this centralized approach when applied to Grids. In these highly distributed environments, there are numerous user communities and shared resources, each with distinct security requirements. No single resource broker is likely to be trusted by all of these communities and resources with the necessary information to make decisions. At the extreme, each user may need his or her own resource broker, because only that user has the authorization to gather all of the information necessary to make brokering decisions.

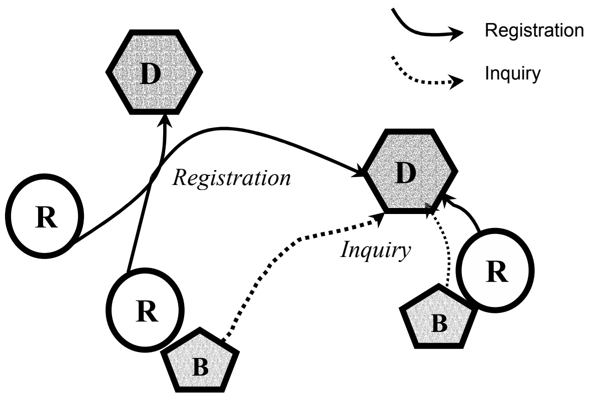


Figure 1: Aggregate directory Services (D), with resources (R), and Broker (B). Depicts resource registration and broker enquiries to directory servers.

Alternatively, we can adopt one of two decentralized models. First and commonly referred as *sender-initialized* scheduling, is an approach wherein entities requiring access to the specific resource initiates and performs the scheduling. Second, is a *receiver-initialized* scheduling wherein entities in possession of the resource perform the selection. The latter approach would require addressing the motivation for a resource owner to volunteer to perform the scheduling. For this reason, we have designed a decentralized storage brokering strategy wherein every client that requires access to a replica performs the selection process rather than a central manager performing matches against clients and replicas (Figure 1).

2.2. Replica Selection Sequence in Globus Data Grid

An application that requires access to replicated data begins by querying an application-specific metadata repository, specifying the characteristics of the desired data (Figure 2). The metadata repository maintains associations between representative characteristics and logical files, thus enabling the application to identify logical files based on application requirements. Once the logical file has been identified, the application uses the replica catalog to locate all replica locations containing physical file instances of this logical file, from which it can choose a suitable instance for retrieval.

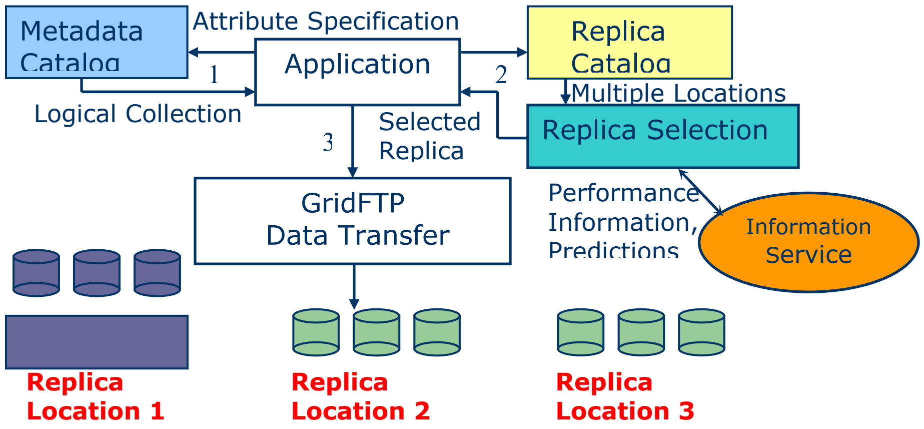


Figure 2: Sequence of event leading to the selection of replicas in the Globus DataGrid environment [Globus02].

The entity that identifies the suitable instance of a replicated file based on application requirements is referred to as a broker. In effect, the responsibility of the broker is to map application requirements against storage resource capabilities obtained from an aggregate information service (Figure 2). In the following section, we discuss the storage broker in detail.

2.3. Storage Broker

Request Specification: Data scheduling requests are represented in terms of classified advertisements, used extensively in Condor for job placement. ClassAds provide a rich environment for resources and jobs to specify their requirements and capabilities in terms of attribute-value pairs. Further, it provides efficient matching and ranking mechanisms. An application might advertise its request to the broker as follows:

```
        hostname = "comet.xyz.com";
        reqdRDBandwidth = 5M/sec;
        rank = other.AvgRDBandwidth;
        requirement = other.AvgRDBandwidth > 5M/sec;
```

The application indicates its preference for a storage resource that offers a maximum transfer bandwidth greater than 5 MB/sec.

Discovery: Information-discovery primarily involves searching for relevant details and capabilities of resources from multiple administrative domains by querying aggregate directory servers (Grid Index Information Servers, GIISs), catalogs, etc. The GIIS typically caches/obtains data in the form Lightweight Directory Access Protocol (LDAP) objectClasses that are essentially groupings of data represented in terms attribute-value pairs. Examples of objectClasses include information on resources such as: cpu, storage, networks, etc., that can be obtained from specific information prociders. Searches can further be strengthened by directing more specific queries to individual resources (Grid Resource Information Servers, GRISs) based on a subset of matched resources, for up-to-date dynamic information. The following is a simple ClassAd describing the capabilities of a storage resource:

```
        hostname = "dpsslx04.lbl.gov";
        volume = "/dev/sandbox";
        availableSpace = 50G;
        AvgRDBandwidth = 6M/sec;
        requirement = true;
```

This ClassAd describes a volume of a storage resource by specifying its attributes. The ClassAd can also specify usage policy enforced by the resource using the "requirement" attribute. When two ClassAds are being matched, a MatchClassAd is created that contains both ClassAds. Each ClassAd can refer to the other ClassAd by using the "other" keyword. We will discuss the storage capability objectClass in more detail in subsequent sections.

Matching, Ranking, Action: Information obtained through discovery is in the form of LDAP objectClasses, while requests are in ClassAds. Search results are converted to ClassAds and then matched and ranked against requirement specification. A list of resources, once established, can then be used to fetch data from, using GridFTP.

2.4. Data Access Service

In the previous sections we described our decentralized architecture for replica selection. But, intelligent replica selection requires information about the capabilities and performance characteristics of storage systems (replica locations). Broker decisions are only as good as the information they have access to. In the snippet ClassAds above, we used "`avgRDbandwidth`" as a metric to decide from among a set of replica locations. GridFTP is the tool widely used to move data in the Grid and the `avgRDbandwidth` is the average GridFTP bandwidth between the site hosting the replica and the site requesting it. It is such predictive information we are interested in for brokering. In this section, we briefly highlight how such storage system functionality can be exposed in the context of an information service so it can be discovered by a broker.

We are particularly interested in the speed of a storage system, or rather, the time that the storage system will take to deliver a replica. One approach to determining this information is to construct a performance model of the relevant components (e.g., see [SC00]). We favor an alternative approach in which historical information concerning GridFTP data transfer rates is used as a predictor of future transfer times. In brief, storage systems are configured to provide information on their own behavior and performance. Attributes such as maximum achievable read and write transfer bandwidths across networks can help an application choose one replica over another. Storage replicas monitor their own performance to gather and publish attribute data. This feature can be extended further, to obtain statistical information based on the performance data, such as average transfer bandwidths and their standard deviations that can help predict the behavior of a particular replica. Further, we expect that there will be significant reuse of storage servers by clients, thereby justifying performance information on a per source basis, which provides a client useful information on end-to-end transfer performance. For example, a simple heuristic of combining past observed performance with current load of server might give a client a reasonably good choice of server.

We instrumented the GridFTP server by adding mechanisms to log performance information for every file transfer. Log entries include source address, file name, file size, number of parallel streams, TCP buffer size for the transfer, start and end timestamps, total time consumed by the transfer, aggregate bandwidth achieved for the transfer, nature of the operation (read or write), and logical volume to and from which file was transferred. For the GridFTP monitoring data, we built an information provider that accesses the log data to advertise a set of recent measurements as well as some summary statistic data. To generate statistical information on transfers, we developed LDAP shell-backend scripts to filter the information in the logs. In addition, we developed schemas for this data. Figure 3 presents a fragment of the output from a GridFTP information provider (details include: prediction information, GridFTP server and port information, etc.). Combined, these enable a GridFTP performance information provider to process logs by building schemas and scripts to publish statistical information. Replica locations (sites running GridFTP servers) publish such performance information using GRIS servers.

GridFTP Information Provider Output

```
dn:"140.221.65.69,
hostname=dpsslx04.lbl.gov,dc=lbl,dc=gov,o=grid"
cn:"140.221.65.69"
hostname:"dpsslx04.lbl.gov"
gridftpurl:"gsiftp://dpsslx04.lbl.gov:61000"
minrdbandwidth:1462K
maxrdbandwidth:12800K
avgrdbandwidth:6000K
avgrdbandwidthtenmbrange:5714K
```

Figure 3: A fragment of the output from the GridFTP performance information provider registered with the GRIS at LBL.

3.0. Predicting Bulk Data Transfers

In the following sections we discuss statistical techniques to synthesize bulk data transfer forecasts which can then be used to differentiate the various replica locations. Our goal is to obtain accurate predictions of file transfer times between a storage system and a client. Achieving this can be challenging because numerous devices are involved in the end-to-end path between the source and the client, and the performance of each (shared) device along the end-to-end path may vary in unpredictable ways.

One approach to predicting this information is to construct performance models for each system component (CPUs at the level of cache hits and disk access, networks at the level of the individual routers, etc.) and then to use these models to determine a schedule for all data transfers [SC00], similar to classical scheduling [Adve93, Cole89, CQ93, Crovella99, ML90, Schopf97, TB86, ZLP96]. In practice, however, it is often unclear how to combine this data to achieve accurate end-to-end measurements. Also, since system components are shared, their behavior can vary in unpredictable ways [SB98]. Further, modeling individual components in a system may not capture the significant effects that these components have on each other, thereby leading to inaccuracies [GT99].

Alternatively, observations from past application performance of the entire system can be used to predict end-to-end behavior. The use of whole-system observation has relevant properties for our purposes. These predictions can, in principle, capture both evolution in system configuration and temporal patterns in load. A by-product of capturing entire system evolution is enhanced transparency, in that we can construct such predictions without detailed knowledge of the underlying physical devices. This technique is used by Downey [Downey97] and Smith et al. [SFT98] to predict queue wait times and by numerous tools (Network Weather Service [Wolski98], NetLogger [NetLogger02], Web100 [Web100Project02], iperf [TF01], and Netperf [Jones02]) to predict the network behavior of small file transfers. We adopted the use of end-to-end GridFTP measurements, capturing whole-system behavior, obtained through the instrumentation process described earlier to derive predictions.

3.1. Univariate Predictors

In this section we describe some of the predictors we developed, categorize possible approaches into mean-based, median-based, and autoregressive techniques. These predictors are applied to a single variable, namely the GridFTP transfer logs between any site pair. We use several variations of each of these models in our experiments.

Mean-based, or averaging, techniques are a standard class of predictors that use arithmetic averaging (as an estimate of the mean value) over some portion of the measurement history to estimate future behavior. The general formula for these techniques is the sum of the previous *n* values over the number of measurements. Mean-based predictors are easy to implement and impose minimally on system resources.

A second class of standard predictors is based on evaluating the median of a set of values. Given an ordered list of *t* values, if *t* is odd, the median is the (*t*+1)/2 value; if *t* is even, the median is half of the *t*/2 value added with the (*t*+1)/2 value. Median-based predictors are particularly useful if the measurements contain randomly occurring asymmetric outliers that are rejected. However, they lack some of the smoothing that occurs with a mean-based method, possibly resulting in forecasts with a considerable amount of jitter [HP91].

	Average based	**Median based**	**Autoregression**
All data	AVG	MED	AR
Last 1 Value	LV		
Last 5 Values	AVG5	MED5	
Last 15 Values	AVG15	MED15	
Last 25 Values	AVG25	MED25	
Last 5 Hours	AVG5hr		
Last 15 Hours	AVG15hr		
Last 25 Hours	AVG25hr		
Last 5 Days			AR5d
Last 10 Days			AR10d

Figure 4: Univariate predictors.

The third class of common predictors is autoregressive models [GP94, HP91, Wolski98]. We use an autoregressive integrated moving average (ARIMA) model technique that is constructed using the equation

$$G' = a + bG_{t-1},$$

where G' is the GridFTP prediction for time, t, G_{t-1} is the previous data occurrence, and *a* and *b* are the regression coefficients that are computed based on past occurrences of G using the method of least squares. This approach is most appropriate when there are at least fifty measurements and the data is measured with equally spaced time intervals. Our data does not meet these constraints, but we include this technique to do a full comparison. The main advantage of using an ARIMA model is that it gives a weighted average of the past values of the series, thereby possibly giving a more accurate prediction. However, in addition to requiring a larger data set than the other techniques to achieve a statistically significant result, the model can have a much greater computational cost.

Further to this, we use measurement and temporal windows that limit the datasets based on number of values or on time. Filtering in this way is based on the assumption that recent measurements are more reflective of current system behavior. Figure 4 summarizes our various predictors.

3.2. Multivariate Predictors

The obvious downside of univariate predictors has nothing to do with the predictors themselves but more so with the nature of data transfers on the Grid. Because of the sporadic nature of transfers, predictors based on log data alone may fail to factor in current system trends and fluctuations. To mitigate the adverse effects of this problem, we introduce other periodic datastreams to expose the behavior of components in the end-to-end data path and to reveal the current environment on the Grid. The working hypothesis here is to capture the ambience in which the GridFTP transfers are performed so we can use it to better explain future behavior. From our preliminary assessment we found that networks and disks contributed up to 70% and 30% respectively of the actual transfer time.

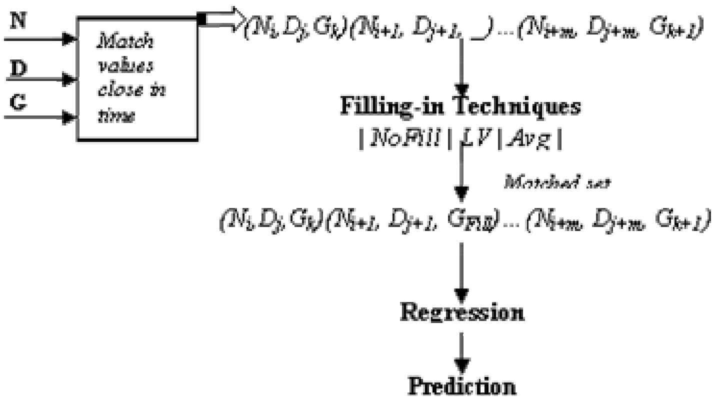

N
D
G
Match values close in time
$(N_i, D_j, G_k)(N_{i+1}, D_{j+1}, \_)\ldots(N_{i+m}, D_{j+m}, G_{k+1})$
Filling-in Techniques
| NoFill | LV | Avg |
Matched set
$(N_i, D_j, G_k)(N_{i+1}, D_{j+1}, G_{Fill})\ldots(N_{i+m}, D_{j+m}, G_{k+1})$
Regression
Prediction

Figure 5: Sequence of events for deriving predictions from GridFTP (G), disk load (D), and NWS (N) datastreams.

We used tools such as Network Weather Service (NWS) and iostat to periodically monitor and log network and disk behavior respectively in the form of <timestamp, load-value>. From our studies we also found that NWS and iostat probes were correlated to the eventual GridFTP throughput achieved (up to 0.7 correlation value). Based on this, we developed a set of multivariate predictors using regression models to predict from a combination of several data sources – GridFTP log data and network load data, GridFTP log data and disk load data, or a combination of all three. The datastreams require some preprocessing before the regression techniques can be applied to them. This includes time matching the data streams and filling-in techniques.

3.2.1 Matching

Our three data sources (GridFTP, disk I/O, and NWS network data) are collected exclusive of each other and rarely have the same timestamps. To use regressive models on the data streams, however, we need to have a one-to-one mapping for the values in each stream. Hence, we are required to match values from the three sets such that for each GridFTP value, we find disk I/O and network observations that were made around the same time.

For each GridFTP data point (T_G, G), we match a corresponding disk load (T_D, D) and NWS data point (T_N, N) such that T_N and T_D are the closest to T_G. By doing this, the triplet (N_i, D_j, G_k) represents an observed end-to-end GridFTP throughput (G_k) resulting from a data transfer that occurred with the disk load (D_j) and network probe value (N_i) backdrop.

At the end of the matching process, the three datastreams have been combined into the sequence that looks like

$$(N_i, D_j, G_k)\ (N_{i+1}, D_{j+1}, \_)\ldots(N_{i+m}, D_{j+m}, G_{k+1}),$$

where G_k, and G_{k+1} are two successive GridFTP file transfers, N_i and N_{i+m} are NWS measurements, and D_j and D_{j+m} are disk load values that occurred in the same timeframe as the two GridFTP transfers. The sequence also consists of a number of disk load and NWS measurements between the two transfers for which there are no equivalent GridFTP values, such as $(N_{i+1}, D_{j+1}, \_)$. Note that these interspersed network and disk load values are time-aligned. Also note that we have described the matching process with reference to all three data sources. In the case where a prediction uses a different number of datastreams, similar matching techniques can be employed.

3.2.2. Filling-in Techniques

After matching the datastreams, we need to address the tuples that do not have values for the GridFTP data – that is, the NWS data or disk I/O data collected in between the sporadic GridFTP transfers. Regression models expect a one-to-one mapping between the data values, so we can either discard the network and I/O data for which there are no equivalent GridFTP data (our NoFill technique) or fill in synthetic transfer values using either an average over the past day's data (Avg), or the last value (LV). Once filled in, the sequence is as follows:

$$(N_i, D_j, G_k)\ (N_{i+1}, D_{j+1}, G_{Fill})\ldots(N_{i+m}, D_{j+m}, G_{k+1})$$

where G_{Fill} is the synthetic GridFTP value. Data, once matched and filled in, is fed to regression models (Figure 5).

3.2.3. Regression Models

Simple linear regression attempts to build linear models between dependent and independent variables. The following equation builds linear models between several independent variables $N_1, N_2, ..., N_k$ and dependent variable G as follows:

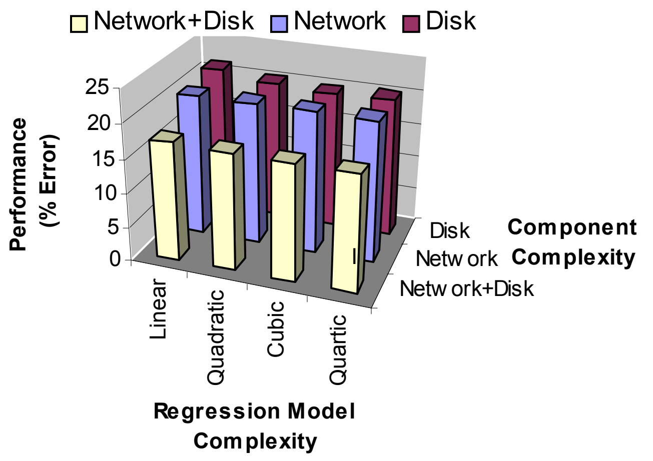


Figure 6: Visualization comparing error, complexity of algorithm, and components included for the site pair LBL and ANL.

$$G' = a + b_1N_1 + b_2N_2 + ... + b_kN_k,$$

where G' is the prediction of the observed value of G for the corresponding values of $N_1, N_2, ..., N_k$. The coefficients a, b_1, b_2, and b_k are calculated by using the method of least squares [Edwards84]. For our case, we built linear models between NWS (N), disk (D), and GridFTP (G) data as explained above, with N and D as independent variables. To improve prediction accuracy, we also developed a set of nonlinear models adding polynomial terms to the linear equation. For instance, a quadratic model is as follows:

$$G' = a + b_1N + b_2N^2.$$

Cubic and quartic models have additional terms b_3N^3 and b_4N^4, respectively. Similar to the linear model, the coefficients in quadratic, cubic, and quartic models b_2, b_3, and b_4 are computed by using the method of least squares. Adding polynomial terms to the regression model can reach a saturation point (no significant improvement in prediction accuracy observed), suggesting that a particular model sufficiently captures the relationship between the two variables [OM88, Pankratz91]. Figure 6 shows a bar graph that compares error, complexity of algorithm, and components included for the site pair, Lawrence Berkeley and Argonne National Laboratories.

3.3. Predictor Performance

We evaluated the performance of our regression techniques on datasets collected over three distinct two-week durations: August 2001, December 2001, and January 2002. In the following subsections we describe the experimental setup, prediction error calculations, and the results obtained from these datasets.

3.3.1. Experimental Setup

The experiments we ran consisted of controlled GridFTP transfers, NWS (64KB probes every five minutes) network sensor measurements, and disk throughput monitoring (every five minutes) between four sites in our testbed (Figure 7): Argonne National Laboratory (ANL), the University of Southern California Information Sciences Institute (ISI), Lawrence Berkeley National Laboratory (LBL), University of Florida at Gainesville (UFL), and Boston University (BU). All our sites comprised of 100 Mb/sec Ethernets with high-end storage.

GridFTP experiments included transfers comprising several file sizes ranging from 10 MB to 1 GB, performed at random time intervals within 12-hour periods. We calculated buffer sizes by using the formula

*RTT * "bottleneck bandwidth in the link"*

with roundtrip times (RTT) values obtained from ping and with bottleneck bandwidth obtained by using iperf [TF01]. Figure 7 shows the roundtrip times and bottleneck bandwidth for our site pairs. Our GridFTP experiments were performed with tuned TCP buffer settings (1 MB based on the bandwidth delay product) and eight parallel streams to achieve enhanced throughput. Logs of these transfers were maintained at the respective sites and can be found at [Traces02]. For each data set and predictor, we used a 15-value training set; that is, we assumed that at the start of a predictive technique there were at least 15 GridFTP values in the log file (approximately two days worth of data).

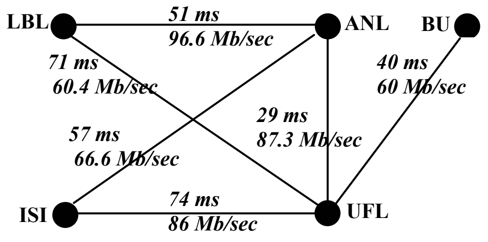


Figure 7: Network settings for our testbed sites. All sites are connected through OC-12 or OC-48 network links. For each site pair round trip times and network bottleneck bandwidths for the link between them is shown.

3.3.2. Metrics

We calculate the prediction accuracy using the normalized percentage error calculation:

$$\%\,Error = \frac{\sum |\,Measured_{BW} - Predicted_{BW}\,|}{(size * Mean_{BW})} * 100,$$

where *size* is the total number of predictions and the Mean is the average measured GridFTP throughput. In this subsection we show results based on the August 2001 dataset Tables 4. Complete results can be found in [VS03].

In addition to evaluating the error of our predictions, we evaluate information about the variance in the error. Depending on the use case, a user may be more interested in selecting a site that has reasonable performance bandwidth estimates with a relatively low prediction error than in selecting a resource with higher performance estimates and a possibly much higher error in prediction. In such cases, it can be useful if the forecasting error can be stated with some confidence and with a maximum/minimum variation range. These limits can also, in theory, be used as catalysts for corrective measures in case of performance degradation.

In our case, we can also use these limits to verify the inherent cost of accuracy of the predictors. By comparing the confidence intervals of these prediction error rates, we can determine whether the accuracy achieved is at the cost of greater variability, in which case there is little gain in increasing the component complexity of our prediction approach. Thus, for any predictor (for any site pair and a given dataset), the information denoted by the following triplet can be used as a metric to gauge its accuracy:

Accuracy-Metric = [PredictedThroughput, AvgPast % Error-Rate, ConfidenceLimit],

where *PredictedThroughput* is the predicted GridFTP value (higher the better), with a certain percentage prediction error (the lower the better) and a percentage confidence interval for the error (the smaller the better).

3.3.3. Univariate Predictor Performance

The major result from these predictions is that even simple techniques have a worst-case prediction of about 25%, quite respectable for pragmatic prediction systems. Figure 8 shows the result of sorting the data by file size, since GridFTP throughput varied with transfer file sizes. We grouped several file sizes into categories: 0–50 MB as 10M, 50–250 MB as 100M, 250–750 MB as 500M, and more than 750 MB as 1G, based on the achievable bandwidth. We observe almost up to 10% increase in accuracy with context sensitive filtering.

In general, for our univariate predictors, we did not see a noticeable advantage of limiting either average or median techniques using a sliding window or time frames. The ARIMA models did not see improved performance for our data, although they are significantly more expensive compared to simple means and medians. This is likely due to the irregular nature of our data. Average and median based predictors (and their temporal variants) for a GridFTP dataset size of 50 values was computed under a millisecond, while autoregression on the same set consumed a few milliseconds.

3.3.4. Multivariate Predictor Performance

Table 4 shows the performance gains of using a regression prediction with GridFTP and NWS network data (G+N) over using the GridFTP log data univariate predictor in isolation (first two shaded columns in the table). We use the moving average (AVG25) as a representative of univariate predictor performance. For our datasets, we observed a 4% to 6% improvement in prediction accuracy when the regression techniques with LV or AVG filling were used. Regression with NoFill (throwing away the unmatched GridFTP data) shows no significant improvement when compared with univariate predictors.

Table 4 also shows that including disk I/O component load variations in the regression model provides us with gains of 2% to 4% (G+D Avg) when compared with AVG25 (first and third shaded columns in the table). Different filling techniques (G+D Avg and G+D LV) perform similarly, and again NoFill shows no improvement, or even a decrease in accuracy, when compared with univariate predictors.

Table 1: Normalized percent prediction error rates for the testbed site pairs for the August 2001 dataset. The figure denotes four categories: (1) prediction based on GridFTP data in isolation (AVG25), (2) regression between GridFTP and NWS network data with the three filling in techniques (G+N), (3) regression between GridFTP and disk I/O data with the three filling in techniques (G+D), and (4) regression based on all three data sources (G+N+D). Shaded portions indicate a "best of class" comparison between the approaches. All percentage values are averages based on different file categories.

	Only GidFTP Logs [VSF02]	**Linear Regression between GridFTP Logs and Network Load [VS02]**			**Linear Regression between GridFTP Logs and Disk Load**			**Linear Regression using all Three Data Sources**		
	AVG25	**G+N NoFill**	**G+N LV**	**G+N Avg**	**G+D NoFill**	**G+D LV**	**G+D Avg**	**G+N+D NoFill**	**G+N+D LV**	**G+N+D Avg**
LBL-ANL	24.4%	22.4%	20.6%	20%	25.2%	21.7%	21.4%	22.3%	17.7%	17.5%
LBL-UFL	15%	18.8%	11.1%	11%	20.1%	11.6%	11.9%	11.1%	8.7%	8%
ISI-ANL	15%	12%	9.5%	9%	13.1%	13%	11.4%	11%	8.9%	8.3%
ISI-UFL	21%	21.9%	16%	14.5%	22.7%	19.7%	18.8%	14.7%	13%	12%
ANL-UFL	20%	21%	20%	16%	21.8%	19.9%	19.3%	15.3%	16.7%	15.5%

Comparing the second and third block of data in Table 4 shows that all variations of predictors using NWS data (G+N) perform better than predictors using disk I/O data (G+D) in general. This observation agrees with our initial measurements that only 15% to 30% of the total transfer time is spent in I/O, while the majority of the transfer time (in our experiments) is spent performing network transport.

When we include both disk I/O and NWS network data in the regression model (G+N+D) along with GridFTP transfer logs, we see prediction error drop of 8% to 17% and up to 3% improvement when compared with G+N (second and fourth shaded columns in Table 4) and a 6% improvement over G+D (third and fourth shaded columns in Table 4). Overall, we see up to 9% improvement when we compare G+N+D with the original univariate prediction based on AVG25.

Figure 9a compares the average prediction error for Moving Avg, G+D Avg, G+N Avg, and G+N+D Avg for all of our site pairs (represents the shaded columns in Table 4) and also presents 95% confidence limits for our prediction error rates. The prediction accuracy trend is as follows:

Moving Avg < (G+D Avg) < (G+N Avg) < (G+N+D Avg)

Figure 9b shows that the confidence interval (the variance in the error) does in fact reduce with more accurate predictors, but the reduction is not significant for our datasets. We observed no noticeable improvements in using polynomial models for our experiments.

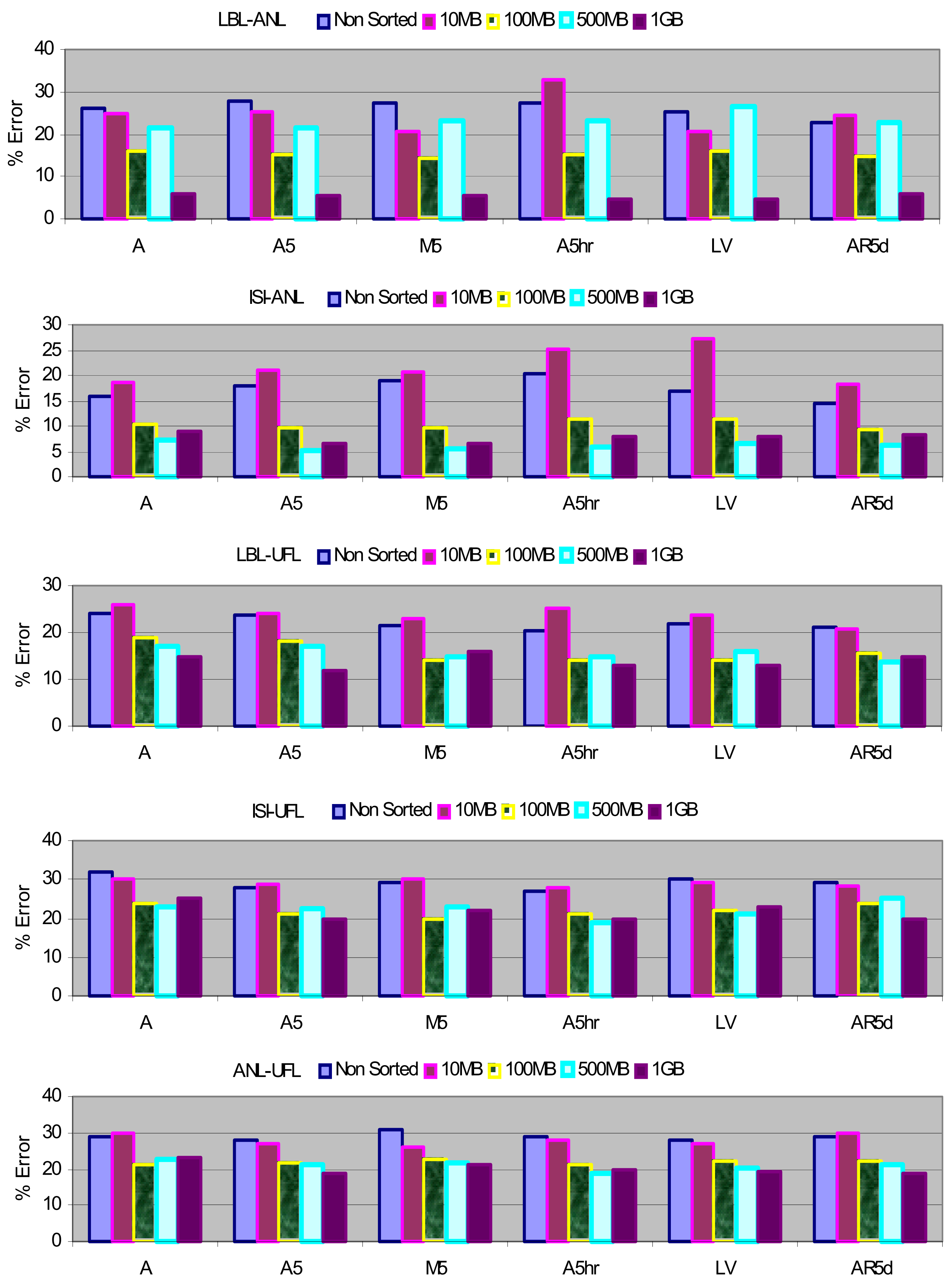
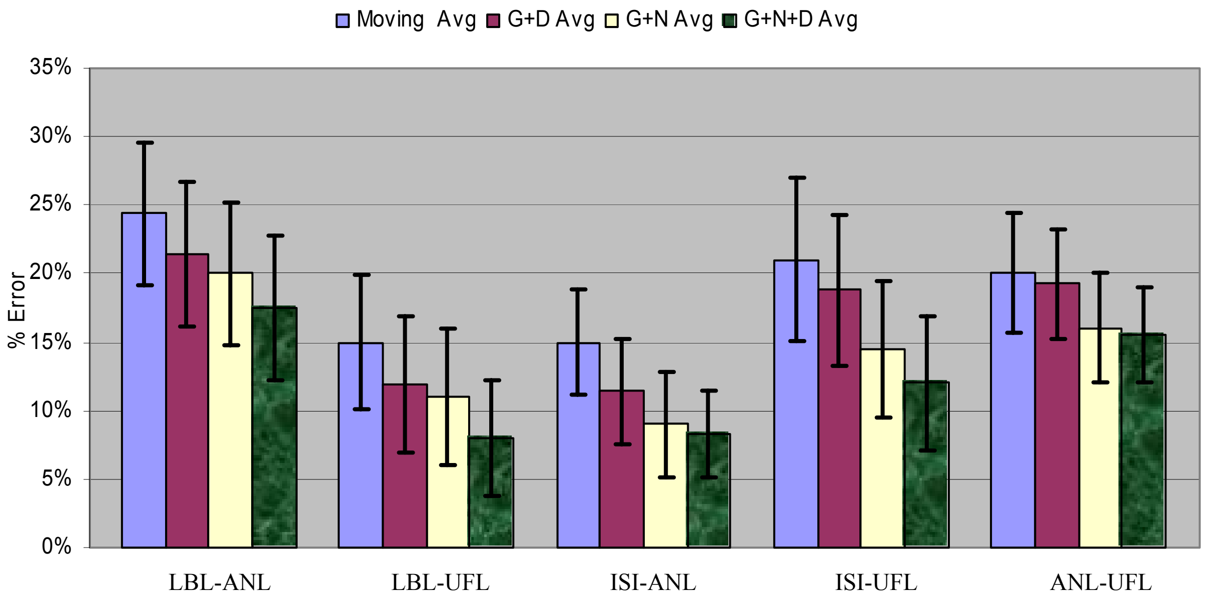
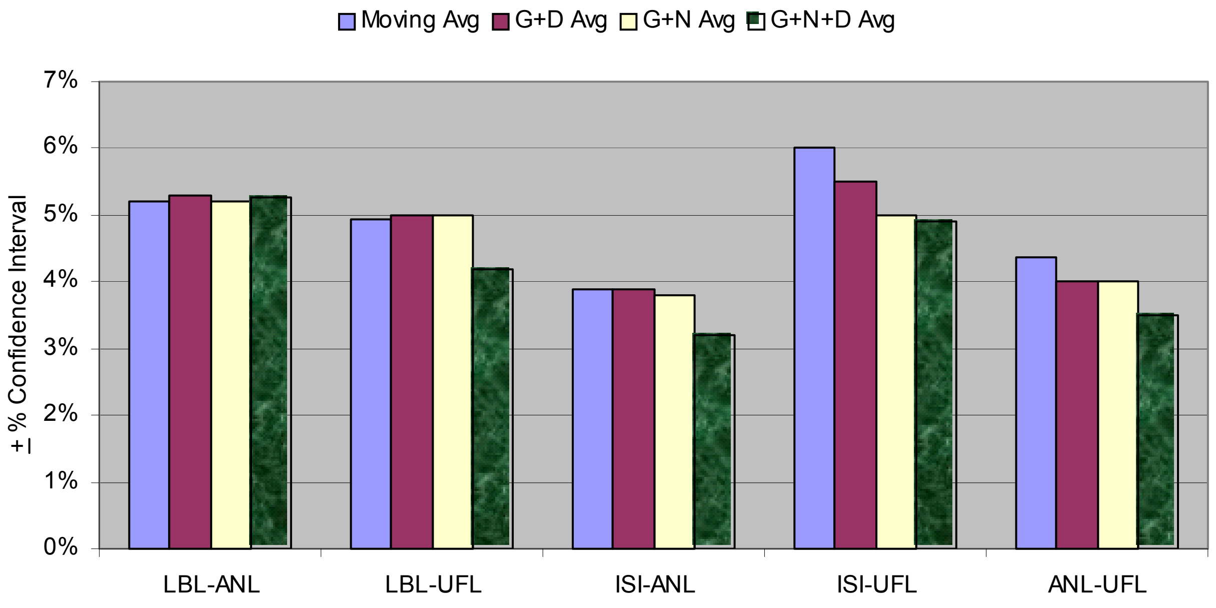


Figure 8: Impact of classification and the reduction in percent error rates for the testbed (context-sensitive filtering).



(a) Comparison of normalized percent errors for the predictors with 95% confidence limits



(b) Comparison of intervals for the predictors

Figure 9: (a) Normalized percent prediction error and 95% confidence limits for August 2001 dataset based on (1) prediction based on GridFTP in isolation (MovingAvg), (2) regression between GridFTP and disk I/O with Avg filling strategy (G+D Avg); (3) regression between GridFTP and NWS network data with Avg filling strategy (G+N Avg), and (4) regressing all three datasets (G+N+D Avg). Confidence Limits denote the upper and lower bounds of prediction error. For instance, the LBL-ANL pair had a prediction range of [17.3% ± 5.2%]. (b) Comparison of the percentage of variability among the predictors.

4.0. Co-Allocating Data Transfers

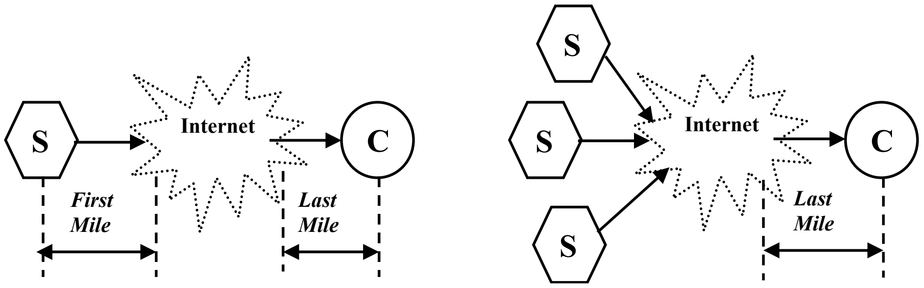
Replica locations once identified and ranked using our selection (Section 2) and prediction (Section3) heuristics can then be used to deliver the dataset collectively to the client. Due to the varied performance characteristics in the replica locations and the links connecting them to the client, downloading large datasets (10 MB – 1 GB) can result in varied end-user experience.

A typical Internet download between a client and a server is mired by several bottlenecks (Figure 10a) [Akamai00]. First, the bandwidth achievable by the client is limited by the bandwidth of the server's connection to the Internet – commonly referred as the *First-Mile* problem. The first-mile bottleneck is further compounded by simultaneous requests to a server from multiple clients. Second, the achievable bandwidth is further limited by the congestion in the link connecting the server and the client. Third, the bottleneck could be in the client's own connectivity to the Internet – the *Last-Mile*. Thus, the download speed is only as fast as the slowest link in the aforementioned setup. Sophisticated solutions are required to significantly address this issue.

One way to improve download speeds is to employ complex server selection techniques to determine the best replica location, offering high transfer rates, using a combination of server and network load details [Akamai02, VTF01]. In practice, however, due to the shared nature of network links the load on them can vary unpredictably. Thus, in the face of transient network conditions, downloading datasets even from the best of servers can often result in pedestrian transfer rates.

A promising alternative is to download data from multiple locations, establishing multiple connections in parallel (Figure 10b). With this approach, instead of downloading the entire dataset from a single sever, unique partial copies of the dataset are fetched from multiple servers in parallel that are later reassembled at the client end.

This co-allocation of data transfers has several relevant properties of significant interest to us. First, it obviates the need for complex server selection. Second, due to its decentralized nature the eventual performance achieved may not be adversely affected by degradation in any of the co-allocated flows while also being resilient to server failures. Third, the client download experience can be positively amplified with the aggregate bandwidth commensurate to the summation of the individual transfer rates of each flow. Fourth, it significantly alleviates the first-mile (slow server, serving a fast client) and the Internet congestion problem by distributing load to multiple servers and different routes (Figure 25b). Even in the case of a slow client served by a fast server, co-allocation can offer significant benefits due to fluctuations in network conditions.



(a) Typical Internet Download **(b) Co-Allocated Download**

Figure 10: (a) Various bottlenecks in the Internet document download – the first mile problem, the congestion in the links connecting server and client and the last mile problem. (b) Co-Allocated download model minimizes the first mile and the link congestion bottlenecks.

Co-allocating data transfers across multiple replica locations can have widespread applicability beyond scientific data-sharing communities. For instance, Internet content providers rely heavily on content distribution networks [JCD+00] for managing consistent replicas of popular content on surrogate, edge-servers, closer to end-users [Akamai00]. Content distribution networks such as Akamai [Akamai02] and Speedera [Speedera02] improve download speeds by employing techniques such as request redirection [WPP02, KRR00] (to select best servers) to fetch data from less congested links. Thus, parallel-downloading techniques can find significant use in such scenarios.

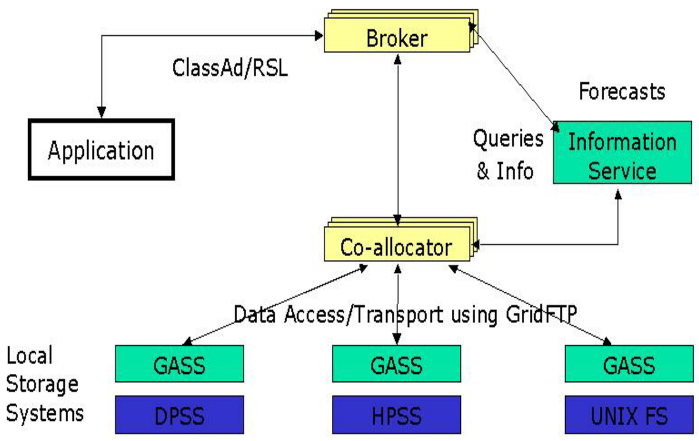


Figure 11: Resource management architecture and the role of co-allocation. Co-allocator combines broker decisions and information services to map data transfer requests onto storage systems using GridFTP and Globus Access to Secondary Storage (GASS).

Peer-to-peer file sharing, where peers come together in a cooperative, decentralized manner to locate and share content, is yet another candidate for parallel downloading [CP02]. The enormous popularity of file sharing means that networked content sharing systems are likely to siphon much of the available Internet bandwidth. In fact, recent studies indicate that file sharing activity contributes up to 60 percent on any service provider network [Sandvine02, SGG02]. Co-allocations in such cases can help improve download speeds, reduce load on certain parts of the network, alleviate loaded peers, etc.

Developing techniques for parallel downloads of Internet documents is of significant interest in the networking community and can be broadly classified into stateless and stateful approaches.

Stateless approaches to the parallel access problem rely on clients subscribing to several mirror sites to restitute the data. This approach further makes extensive use of erasure codes (error correction codes) [Rizzo97] to develop an "n" packet encoding of a "k" packet file, with the property that the file can be reassembled from any "k" packet subset of the encoding [BCM+02, BLM02, BLM99]. Pros of this approach are: obviates the need for maintaining file ranges and renegotiations on a per flow basis, fault tolerance and scalability; while the cons are: constructing an "n" packet encoding is nontrivial, cost of encoding and decoding can be significant for large dataset size, and clients and servers are required to agree, apriori, on common encoding schemes. Other related effort includes Rabin's [Rabin89] and Maxemchuk's [Maxemchuk75] work on dispersing pieces of the file on different nodes in the network for fault tolerance and dispersity routing respectively.

On the contrary, stateful techniques divide the file into disjoint sets, downloading different ranges from different servers. In [RKB00, Gkantsidis02], the authors develop previous history-based and dynamic solutions, demonstrating their techniques for web-based documents of the order of several hundred kilobytes. Accurate predictions of range distributions are required for several stateful techniques, which are often quite difficult to obtain in the face of changing network conditions. In [RKB00, Gkantsidis02], the authors rely on simple averages of previous transfer rates as an estimate for range calculations per flow. Work from Beck et. al., demonstrated the usefulness of dynamic distributed downloads in the context of streaming applications by fetching multiple copies of file blocks in order to address jitter [PAD+02].

In our work we develop history-based and dynamic solutions similar to that of [RKB00, Gkantsidis02, PAD+02], but extend it by addressing network fluctuations. Further, we employ prediction techniques (previous section) to predict data transfer rates between sources and sinks, and use those for range calculations per flow that can significantly improve our performance and reduce renegotiations. The use of encoding schemes, and thus the stateless alternative, may not be suited for our purposes due to our concentration on large datasets, for which encoding and decoding times can be quite significant.

4.1. Co-Allocation Architecture

The Globus Toolkit [FK98] provides a basic template for resource management [CFK99], which can be extended to support the co-allocation of Grid data transfers. As illustrated in Figure 11, the architecture comprises of three main components: an information service, local storage systems, and broker/co-allocator.

An application requiring access to data presents a description of the data to the broker. The broker, in conjunction with information services [CFF+01], identifies possible alternatives from where the dataset in question can be fetched. This set is then presented to the co-allocation agent, which uses a combination of information services and some heuristics to map the data transfer request across multiple replica locations to download the data in parallel using GridFTP.

4.2. Partial Copy in GridFTP

GridFTP extends standard FTP implementations with several features needed in Grid environments, such as security on control and data channels, multiple data channels for parallel streams, partial file transfers, and third party transfers. Of particular interest to us is the ability to fetch partial copies of a file. Partial copy, and several other features, is part of GridFTP's extended retrieve functionality, which is used to request that a retrieve be done with some additional processing on the server. This command is an extensible way of providing server-side data reduction. With partial copy, a section of the file, defined by the starting offset and extent, will be retrieved from the data server.

4.3. Co-Allocation Mechanisms

We now proceed to describe the co-allocation mechanisms that we have developed. Co-allocation is the idea of splitting the data transfer among available servers to improve client perceived throughput.

4.3.1. Brute-Force Co-Allocation

Brute-force co-allocation is a basic scheme that works by dividing the file size equally among available flows. Thus, if the data to be fetched is of size, "S" and there are "n" locations to fetch it from, then this technique assigns to each flow a data block of size, "S/n". With this technique, although all the available servers are utilized, bandwidth differences among the various client-server links are not exploited.

4.3.2. History-based Co-Allocation

To address and exploit transfer rate differences among the various co-allocated flows, we develop a history-based allocation scheme. With this technique, the block size per flow is commensurate to its predicted transfer rate, based on a previous history of GridFTP transfers. Thus, the file-range distribution is based on the predicted merit of the flow. If these predictions are not accurate enough, renegotiations of flow sizes might be necessary as slower links can be assigned larger portions of data, which could weigh heavily on the eventual bandwidth achieved.

In order to obtain accurate predictions of transfer rates for the various links, we derived predictions based upon our previous work on forecasting GridFTP transfers (Section 3). Our previous work delved into deriving accurate predictions in the face of network and system load fluctuations. We developed a series of univariate and multivariate predictors that forecast GridFTP transfers in isolation and in conjunction with network and disk load data respectively. We used extensive regression analysis to predict within 15% error. Logs of previous GridFTP transfers are fed to univariate or multivariate predictors to forecast. For purposes concerning co-allocations, we use a temporal variation of univariate, average predictor (moving average over time).

With the history-based approach, the client divides a file into "n" disjoint blocks, corresponding to "n" servers. Each server, "i", $1 \leq i \leq n$, has a predicted transfer rate of "B_i" to the client. In theory then, the aggregate bandwidth achievable by the client for the entire download is:

$$A = \sum_{i=1}^{i=n} B_i$$

where "B_i" is the predicted bandwidth per flow and "A" is the aggregate bandwidth. Such a speedup can only be achieved when all servers keep sending data until the file is fully received – i.e., all servers being busy at all times during the entire download. In practice, however, the achieved bandwidth is limited due to network congestion in the various flows (resulting in some servers finishing earlier than others) and the client's ability to handle the bandwidth surplus. Thus, the rate-limiting factor could be anyone of the slowest links in the setup – the servers themselves, the links connecting the client and servers, or the client itself.

Assuming the client is capable of handling the bandwidth surplus, range distributions are calculated as follows. For each server "i", $1 \leq i \leq n$, and for a replica size, "S", the block size per flow is:

$$s_i = \frac{B_i}{A} * S$$

where “s_i” is the block size per flow. Thus, the block size per flow is commensurate to its transfer rate and its ratio of contribution to the achievable aggregate bandwidth. Faster servers are assigned to deliver bigger portions of the file, while slower servers are assigned smaller pieces. Ranges of partial transfers can then be formulated based on this and the whole file reassembled at the client as follows:

$$S = \sum_{i=1}^{i=n} s_i$$

The time taken for the entire download is:

$$T = \sum_{i=1}^{i=n} \frac{s_i}{B_i}$$

where “T” is the total download time. In this manner, this scheme addresses the transfer rate differences among the various co-allocated flows.

4.3.3. Dynamic Co-Allocation

Although we have addressed the rate differences in the flows and exploited it to deliver proportionate pieces of the file per flow, we do not address dynamic network variations that can cause degradation in transfer rates. In spite of careful bandwidth estimates per flow, network traffic and system load can cause servers, previously determined as fast or slow, to behave differently. This can significantly affect the download time. Thus, an end-user is typically interested in dynamic rate adaptation – an allocation scheme that can dynamically adapt to changing network conditions.

One way to address this is to monitor the progress of history-based co-allocated flows, to perform corrective measures in case of performance degradation. For instance, if the performance in a particular flow drops below a threshold, the transfer can be migrated to an alternate location or remaining data can be equally distributed among other existing flows.

Although in theory, these are feasible alternatives, in practice, however, such techniques are quite complex to realize for the following reasons. First, we need to add additional dynamic monitoring capability to our data movement protocol to monitor each flow, which can significantly contribute to the overhead. Second, we need criteria to determine performance degradation, which can be difficult due to changing network/system conditions. Third, even if degradation could be determined, corrective measures such as transfer migration or resizing may require significant renegotiation between clients and servers, which can be more costly than the existing decrease in performance.

A promising alternative is the use of dynamic co-allocation. We developed two variations of dynamic co-allocation: (1) Conservative Load Balancing and (2) Aggressive Load Balancing.

Conservative Load Balancing: With this approach, the rate, and thus how much a server delivers, is decided dynamically instead of being based on previous history. The dataset in question is divided into “k” disjoint blocks of equal size and each one of the available servers is assigned to deliver, in parallel, one block initially. Once a server delivers the block, another block is requested and so on, until the entire file is downloaded.

Faster servers and servers connected to the client through less congested or faster links, will deliver quickly, thus serving larger portions of the file when compared to their slower counterparts. Thus, with this technique, the load on the co-allocated flows is automatically adjusted so that congested links and loaded or slower servers are not further burdened.

With this technique, the number of blocks per download can affect the throughput achieved. We could either have a large number of small blocks or a small number of large blocks. We study the effect of using different block counts and sizes in Section 4.

The key to achieving maximum aggregate bandwidth, as stated earlier, is to keep all available servers busy at all times. In the best case, each server is only idle for duration “t”, where “t” is the time elapsed since the server delivered the last block and until it receives a request for a new block. Neglecting client side processing and multiprogramming at both ends, this is roughly equivalent to one round-trip time, which is insignificant compared to the entire download time.

One obvious downside to this approach is the eventuality of waiting on the slowest server to deliver the final block (same as history-based allocation). An alternative is to stop the slowest flow or dynamically resize blocks to fetch the remaining data from the other servers, although we do not employ this technique.

Aggressive Load Balancing: With the previous method, although faster servers deliver quickly, we only fetch one-block size each time around. Similarly, slower servers would again be assigned to deliver blocks. To address these issues, we add the following functionality to our load balancing scheme: (1) progressively increase the amount of data requested from faster servers; and (2) reduce the amount of data requested from slower servers or stop requesting data altogether.

In order to achieve the stated effect, we develop a few heuristics. For each block delivered by each flow, we compute the rate achieved and compare it against the running maximum of all flow rates. If the rate at which a flow delivered the block is greater than the running maximum, we double the block size for that flow and reset the maximum; if it is less, we maintain the one-block size for the flow; and if the rate is significantly less than the maximum, we stop using the flow. Thus, using these techniques, we address dynamic rate changes in the various co-allocated flows.

Table 2: Trace characteristics for Co-Allocation experiments during October and December 2002.

Trace Characteristics	
Server	GridFTP with support for partial copy
Download Client Schemes	
No co-allocation	Base client,
Static	Brute and History
Dynamic	Conservative and Aggressive
When	October and December 2002
Duration	Two weeks in each month
Client Locations	ANL, UFL
File Sizes	10MB, 100MB, 500MB, 1GB
Transfers per file size per co-allocation scheme	50 to 100
Total Transfers (each month)	1000 – 1200
Prediction Strategy for History-based downloads	Moving average (over time) of previous transfers
Block Counts for Dynamic downloads	5, 10 and 15
Number of Parallel Streams per co-allocated flow	8
Tuned TCP buffers per stream	Yes

4.4. Results and Analysis

We evaluated the performance of our co-allocation schemes on data collected over two distinct two-week periods during October and December 2002. In the following sections we describe the experimental setup, traces and our results.

4.4.2. Experiment Setup

Our experiments comprised GridFTP transfers, using our co-allocation clients, between five sites in our testbed (Figure 7). A prerequisite for downloading data from multiple servers is that the various links connecting the client and servers be bottleneck disjoint [RKB00, BLM99]. If the client-server links share the same bottleneck then there can be little improvement due to co-allocation. From Figure 3, it is evident that the various client-server links for our setup are bottleneck-disjoint (bottleneck bandwidths were determined using iperf [TF01]).

We performed wide-area data transfer experiments using the GridFTP data movement tool. Our servers were standard GridFTP available from the Globus 2.0 Toolkit, while our clients included the various co-allocation schemes. Transfers comprised several file sizes ranging from 10 MB to 1 GB. These transfers were performed with tuned TCP buffer settings (calculated using the bandwidth delay product as shown in Figure 3) and eight parallel streams (per co-allocated flow) to achieve enhanced throughput. All our transfers were performed with co-allocation clients at either ANL or UFL. Table 2 summarizes our trace characteristics. We use a mix of fast and slow servers to study the effect therein.

4.4.3. Performance

In this section we discuss the performance of our co-allocation clients based on the data collected during October and December 2002. We evaluate four co-allocation schemes: (1) Brute-Force Co-allocation (Brute), (2) History-based Co-allocation (History), (3) Conservative Load Balancing (Conservative) and (4) Aggressive Load Balancing (Aggressive). For the two load balancing techniques, we study the effect of various block counts (Conservative-5, Conservative-10, Conservative-15, Aggressive-5, Aggrssive-10 and Aggressive-15) on the bandwidth achieved. We compare each co-allocation scheme against the base case of fetching the entire file from a single server and study the bandwidth improvements therein. The bandwidth measures are averages based on two-week's worth of transfers.

Impact of Client-Server Configurations on Downloads

In Figures 12 and 13, we study the effect of slow servers (or links) with similar performance, serving a fast client. We see that all co-allocation schemes perform better than the base case of downloading the entire file from a single server. We observe that load balancing schemes (conservative and aggressive with block counts of 5) perform better than brute-force or history-based co-allocation and load balancing offers almost double the performance (for our experiments) when compared with the base case. In the case of slow servers serving fast clients there is usually residual bandwidth available that goes unused with typical downloads. With a distributed download, this residual bandwidth is utilized to achieve enhanced throughput.

In Figures 14, 15 and 16, we use a mix of slow and fast servers to study its effect on download. We observe that co-allocation schemes are either better (improvements of up to 2 MB/sec) than or comparable to faster servers in isolation. The figures indicate that the gain due to co-allocation is inversely proportional to the performance gap between the servers. In Figure 14, a faster server saturates a client quickly, leaving available little residual bandwidth for other servers. Co-allocation in such cases offers very little improvement. In Figures 15 and 16, as the performance gap between the servers is low, we observe gains due to co-allocation.

Shared Bottlenecks

As mentioned earlier, in order for a distributed download to payoff, there should exist residual network bandwidth from the client to the additional servers. If not, then accessing additional sites will interfere with existing connections and contribute to the congestion. To study the effect of shared bottlenecks we choose two servers each from the LBL and ISI domains respectively. As stated earlier, in our experiments, each flow uses parallel streams and tuned TCP buffers to fully utilize the available bandwidth. Thus, adding another server (with the same bottleneck) interferes with existing connections. In Figure 17, we can see that the bandwidth achieved due to a distributed download is much less than that achieved by individual servers in isolation.

Sensitivity of Schemes towards Parameters

We analyzed the effect of file sizes, number of flows and block counts on the download performance – i.e., threshold values beyond which co-allocation offered gains or saturated. Figures 12 through 16 show that all our co-allocation schemes offer significant performance improvements (when compared with the base case) as the file size increases. For smaller file sizes we see no improvements in using co-allocation using our data movement tool. A low value for the performance ratio, R, where R is:

R = Co-allocation Cost / Total Time to Download,

results in gains due to co-allocation. The cost of co-allocation involves connection establishment, negotiations, reassembly, resizing, etc. For smaller files, this co-allocation cost is high compared to the total download time.

In increasing the number of co-allocated flows (Figures 15 and 16) we observed that for our testbed and client-server configurations, download performance reached saturation at about 3 or 4 flows. The natural question then is "*With how many flows should a transfer be launched?*" While this is subjective to client-server configurations, choosing an appropriate number of flows is vital to the performance achieved. One way to address this would be to start with a subset of servers ranked using our prediction strategies and applying load balancing techniques to this set. This way we can exploit the merits inherent to both static and dynamic models.

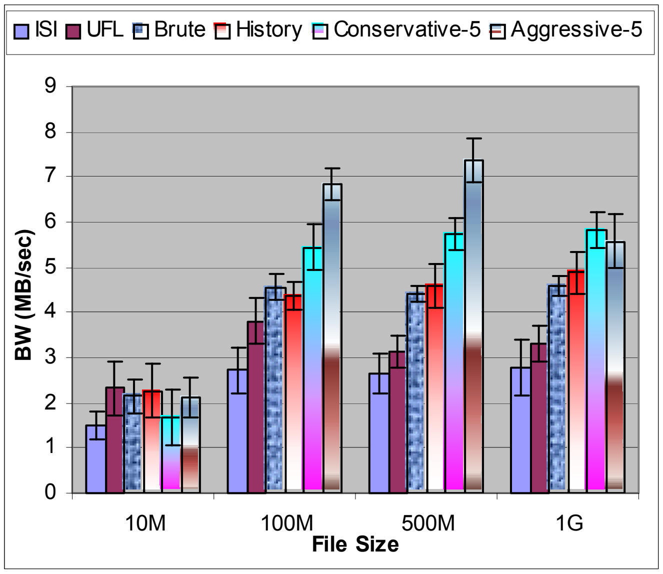


Figure 12: Servers are at ISI and UFL with client at ANL (Oct'02). First two bars in each file size denote downloading the entire file from either ISI or UFL, while others denote co-allocated downloads using the two servers. Depicts 95% confidence ranges for bandwidth.

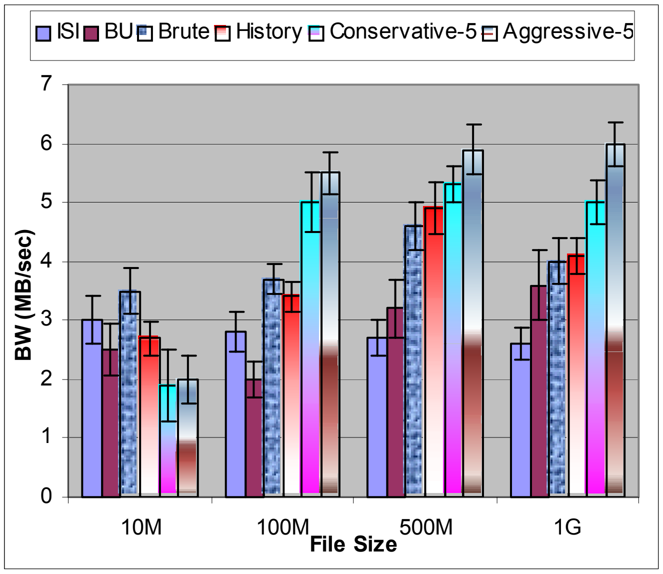


Figure 13: Servers are at ISI and BU with client at UFL (Dec'02). Depicts 95% confidence ranges for bandwidth.

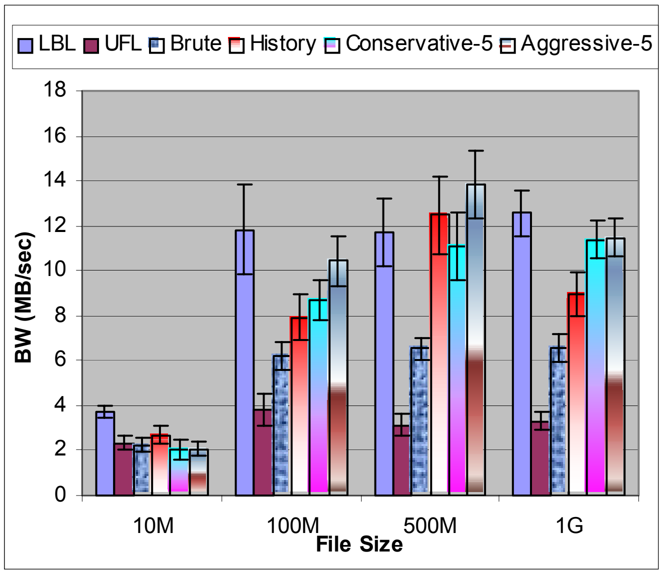


Figure 14: Servers are at LBL and UFL with client at ANL (Oct'02). Depicts 95% confidence ranges for

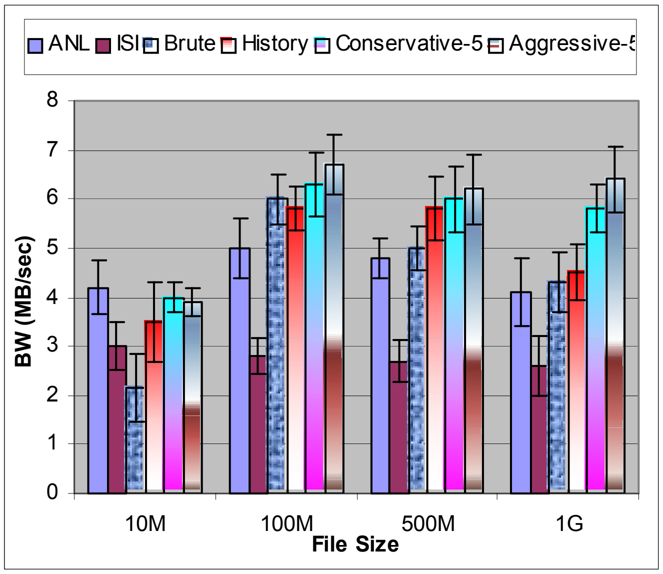


Figure 15: Servers are at ANL and ISI with client at UFL (Dec'02). Depicts 95% confidence ranges for bandwidth.

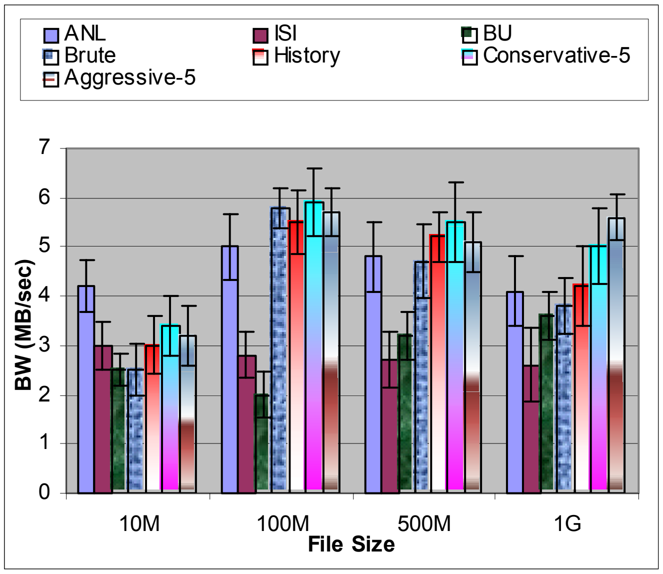
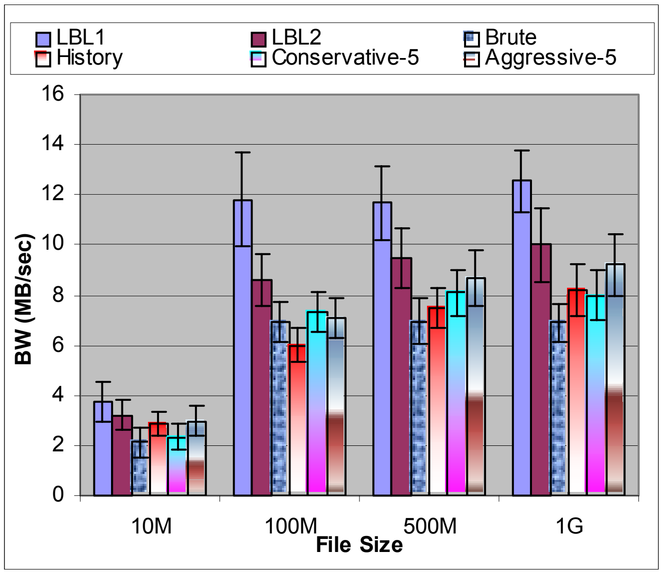


Figure 16: Servers are at ANL, ISI and BU with client at UFL (Dec'02). Depicts 95% confidence ranges for bandwidth.

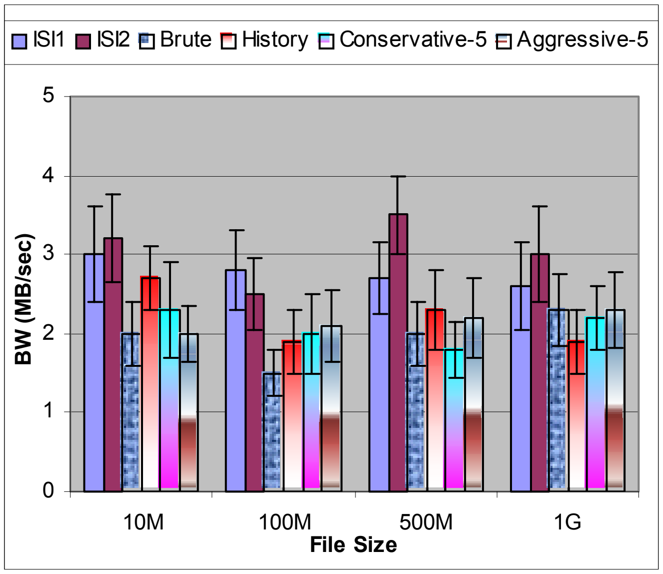
For our various load balancing techniques, we studied the effect of using different block counts (5, 10 and 15). Figure 18 compares the variations of conservative and aggressive load balancing techniques. From the figure we can infer that for smaller file sizes the load balancing schemes perform better with less number of blocks, while for larger file sizes more blocks result in better performance. For our experiments and our block counts we saw performance improvements of up to 1-2 MB/sec. With small files more blocks will result in more overhead in terms of connection establishment, reassembly, etc., when compared to the total download time; while with large files less blocks can mean slower servers delivering bigger portions of the file.

Waiting on Slow Servers

For the load balancing schemes, we analyzed the effect of faster servers waiting on slow servers to deliver the last block. From Figure 19 we can observe that with conservative load balancing (out of the times when slower servers finished last), faster servers are idle for up to 17% of the total download time waiting for slower servers to finish delivering the last block. While aggressive balancing is not altogether devoid of this trend, we observe almost up to 40% reduction in wait times due to a progressive increase in the amount of data fetched from faster servers. The figures also imply that using less number of blocks with larger files results in slower servers having to deliver larger pieces of data, thereby increasing the idle time of faster servers. This suggests that further techniques such as preempting flows or dynamic block sizing, to fetch more from faster servers, are worth investigating.

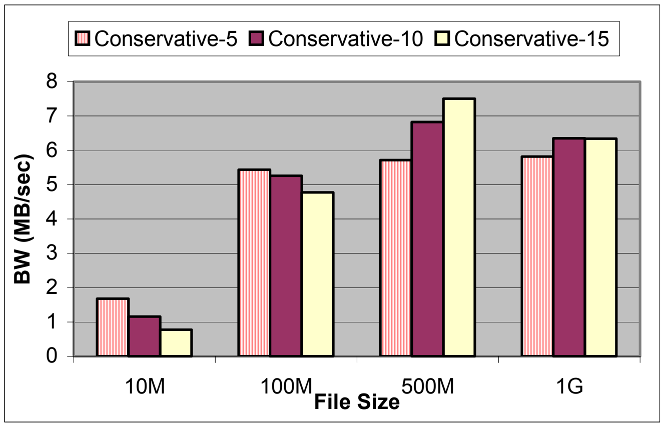


(a) Two servers at LBL sharing the same bottleneck to the client at ANL (Oct'02).

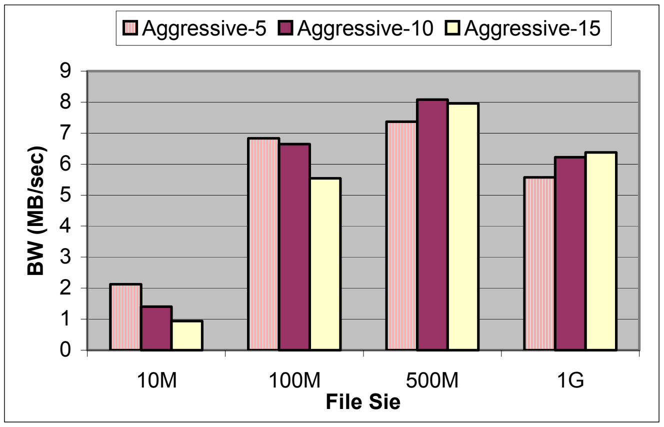


(b) Two servers at ISI sharing the same bottleneck to the client at UFL (Dec'02).

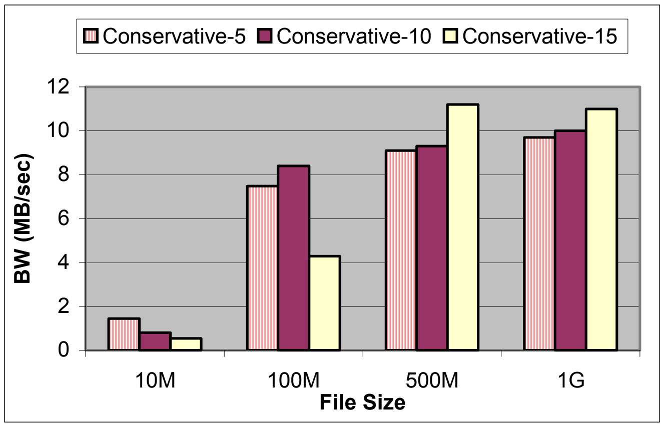
Figure 17: Servers sharing the same network bottleneck to the client. Depicts 95% confidence ranges for bandwidth.



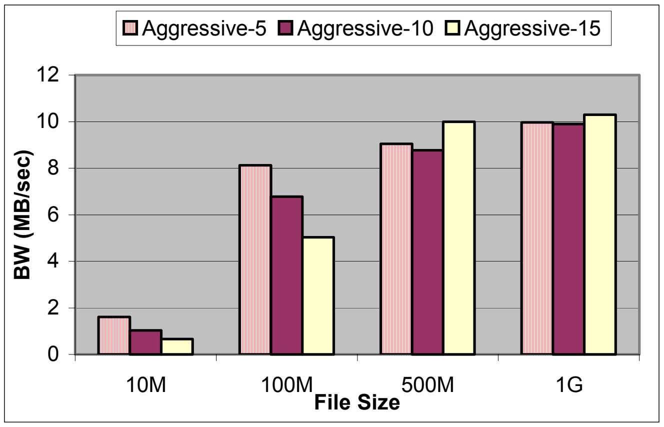
(a) Conservative Load Balancing with servers at ISI and UFL



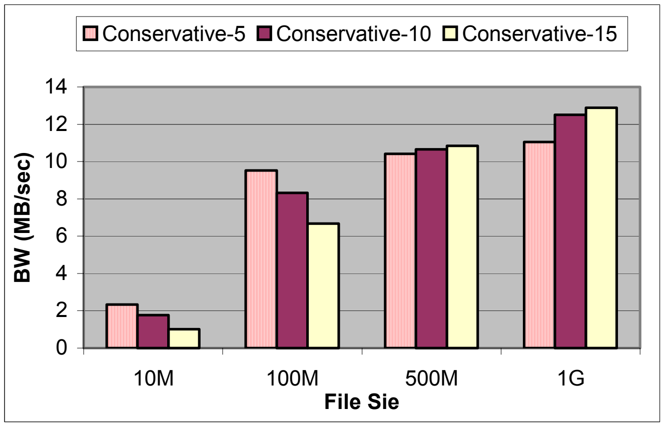
(d) Aggressive Load Balancing with servers at ISI and UFL



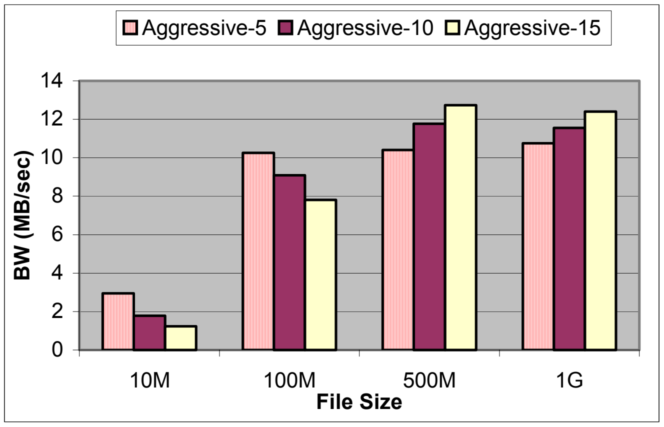
(b) Conservative Load Balancing with servers at LBL and ISI



(e) Aggressive Load Balancing with servers at LBL and ISI

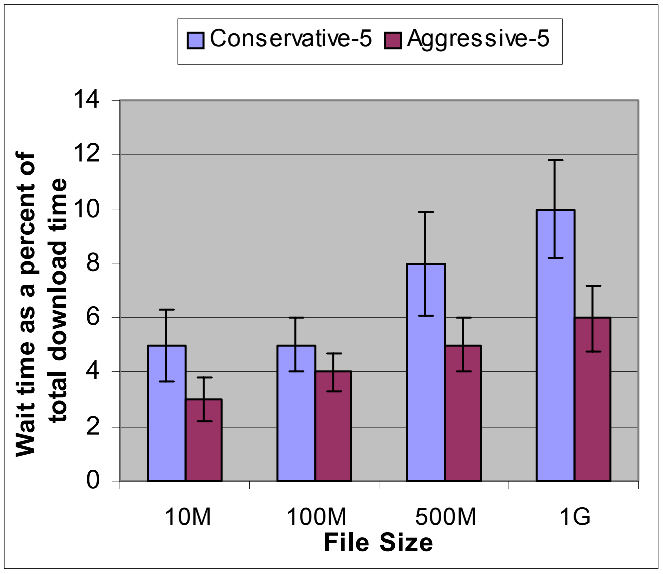


(c) Conservative Load Balancing with servers at LBL, ISI and UFL

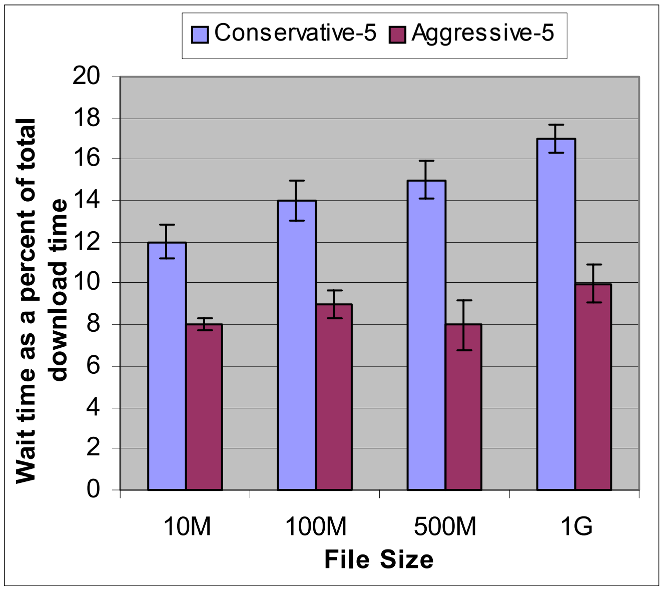


(f) Aggressive Load Balancing with servers at LBL, ISI and UFL

Figure 18: Comparison between the variants of conservative and aggressive load balancing schemes using different block counts for a client at ANL (Oct'02). Conservative-5 denotes a block count of 5.



(a) Servers at ANL and ISI with client at UFL (Dec'02).



(b) Servers at LBL and UFL with client at ANL (Oct'02).

Figure 19: (a) ANL is the faster server having to wait on ISI. (b) LBL is the faster server having to wait on UFL. Bars denote the wait time of the faster server as a percentage of total download time. Also depicts 95% confidence for the % wait times.

5.0. Conclusions and Discussion

In this paper, we have analyzed the orchestration of bulk data movement in Grid environments. We discuss in detail the backdrop of events required to facilitate rapid, efficient and expedited access to massively replicated bulk data. In this regard, we summarize our previous efforts from three fronts, all in the context of the Globus Toolkit. First, we describe a scalable, decentralized replica selection architecture that uses a combination of user specified policies and performance heuristics to locate data from among numerous replicated alternatives. Some of our heuristics include the use of performance estimates of data transfer times, exposed by replica locations, as a key metric of distinction. Second, we develop forecasting machinery to predict future performance rates between sources and sinks. We demonstrate our prediction mechanisms using the GridFTP data movement protocol, progressing from forecasts based on GridFTP transfer logs in isolation to a combination of logs and ambient monitoring data capturing networks and disks. Using these techniques we demonstrate prediction efficacy to be well within 15% error for our testbed sites. Third and finally, we develop co-allocation architecture to use these "selected" and "ranked" sites in unison to download bulk datasets in parallel. With our history based and dynamic load balancing techniques we observe up to almost 2x performance improvements using the GridFTP tool for our sites.

As we alluded before, there is more to bulk data movement than those presented in this paper. In this following discussion, we will briefly highlight some issues. At a very high level, we could classify data management in terms of "*system* challenges" and "*transfer* challenges". By system challenges we refer to all things except the act of data transfer. For instance, this might include ensuring that the dataset in question is available for access which would require the underlying storage system to provide "pinning" abilities; or might require ensuring enough space availability on the remote system for data staging which involves reservation abilities; storage quality of service agreements and guarantees, etc. Other, equally important, issues concern dynamic replica management (replication keeping in mind temporal access patterns) and intelligent scheduling of data transfer requests alongside computation.

Thematic to this paper is the transfer challenge. A bulk of our discussion earlier revolved around transfer-rate estimates and parallel downloading which are key questions guiding several research endeavors. Yet, more needs to be addressed. Tuning of transfer settings is an often used practice in high throughput scientific environments. Bulk data transfers of the order of several gigabytes are usually performed with optimized TCP buffer windows, parallel streams, etc., to achieve enhanced throughput. In fact, all our GridFTP transfer experiments were performed with a 1

MB TCP buffer size and eight parallel streams as opposed to system defaults of 64 KB and a single stream. System defaults are naturally not tuned for the special use case of the high throughput community for reasons concerning fair sharing. Although, with the proliferation of Grid based systems we are quickly faced with the need to provide solutions that exploit the bandwidth explosion. A common rule-of-thumb for buffer tuning is to use "*RTT * Bottleneck Bandwidth*" as an optimal window size between any client-server pair. Tools such as Enable from LBNL extend this further by providing auto-tuning of buffers between any site pair, by exploiting previous history of transfers [TGL+01]. No such service exists for automatically determining the appropriate number of parallel streams for any given site pair. One way to address this issue is to perhaps use previous history of transfers (consisting of bandwidth measures due to various parallel stream counts) between any site pair in conjunction with a network monitoring tool to arrive at educated guesses.

In a related vein, checkpoint file management poses interesting data movement questions. Grid environments consist of multiple sub-jobs with associated data running on different locations. Machines may be reclaimed and might become unavailable in the immediate future. In such cases, checkpoint data or results from computations thus far may have to be moved to the source in an optimal fashion from many locations. Kangaroo, a hop-based checkpoint management system, from Condor addresses some issues therein [TBS+01]. Thus, in order to facilitate efficient bulk data movement, we need to address several of the aforementioned system and transfer challenges.

Acknowledgments

This research was supported in part by fellowships from Argonne National Laboratory and The University of Mississippi. The fellowship from ANL (Summer of 2000 and academic years 2001 and 2002) was due to support by the Mathematics, Information and Computational Sciences Office, Office of Advanced Scientific Computing Research, U. S. Department of Energy, under contract No. W-31-109-Eng-38. The fellowship from The University of Mississippi (Spring 2003) was due to support from the Graduate School's Doctoral Dissertation Award. This research was also supported by the U.S. Department of Energy under contract No. DE-AC05-00OR22725 with UT-Battelle, LLC. We further thank all the system administrators of our testbed sites for their valuable assistance.

Author Biography

Sudharshan S. Vazhkudai is a Research Staff Member at the Computer Science and Mathematics Division of Oak Ridge National Laboratory (ORNL) where he is part of the Network and Cluster Computing Group. His current work revolves around constructing a Distributed Data Grid infrastructure intended to bring massive neutron source data to the TeraGrid. He received his Ph.D. and M.S. in Computer Science from The University of Mississippi in 2003 and 1998 respectively. Prior to that, he received his B.E. in Computer Science from Karnatak University, India in 1996. His doctoral research was on facilitating efficient, rapid access to data in massively distributed Grid environments using user guided selections, transfer-rate predictions and co-allocations. His doctorate was in conjunction with Argonne National Laboratory's Globus Toolkit Project where he was a Research Associate for three years receiving a Givens fellowship (Summer 2000) and a doctoral dissertation fellowship (academic years 2001 and 2002). From 1998 to 2000 he was the project leader of PODOS (Performance Oriented Distributed OS) leading the design and development of a distributed Linux effort, building a high-performance communication architecture, a networked file system and a load sharing environment. His current research interests are distributed resource management in extensible terascale systems, distributed storage management, directory services and peer-to-peer systems.

References

[Akamai00] *Internet Bottlenecks: The Case of Edge Delivery Services*. 2000, Akamai Whitepaper.
[Akamai02] Akamai, http://www.akamai.com, 2002.

[ACF+02] Allcock, W., A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke, *The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets.* Network and Computer Applications, 2002.
[Adve93] Adve, V.S., *Analyzing the Behavior and Performance of Parallel Programs*, in *Department of Computer Science*. 1993, University of Wisconsin.

[AFN+01] Allcock, W., I. Foster, V. Nefedova, A. Chevrenak, E. Deelman, C. Kesselman, A. Sim, A. Shoshani, B. Drach, and D. Williams. *High-Performance Remote Access to Climate Simulation Data: A Challenge Problem for Data Grid Technologies*. in *Supercomputing*. 2001.
[BCM+02] Byers, J.W., et al. Informed Content Delivery Across Overlay Networks. in Proceedings of ACM SIGCOMM'02. 2002.
[BLM99] Byers, J.W., M. Luby, and M. Mitzenmacher. Accessing Multiple Mirror Sites in Parallel: Using Tornado Codes to Speed Up Downloads. in Proceedings of IEEE INFOCOM. 1999.
[BLM02] Byers, J.W., M. Luby, and M. Mitzenmacher, A Digital Fountain Approach to Asynchronous Reliable Multicast. IEEE J-SAC, Special Issue on Network Support for Multicast Communication, 2002. **20**(8): p. 1528-1540.
[BMK96] Basu, S., A. Mukherjee, and S. Kilvansky, *Time Series Models for Internet Traffic*. 1996, Georgia Institute of Technology.
[BMR+98] Baru, C., R. Moore, A. Rajasekar, and M. Wan. *The SDSC Storage Resource Broker*. in *CASCON'98*. 1998.
[CFF+01] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, Grid Information Services for Distributed Resource Sharing, *Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10)*, IEEE Press, August 2001.
[Cole89] Cole, M., *Algorithmic Skeletons: Structured Management of Parallel Computation*. 1989: Pitman/MIT Press.
[CQ93] Clement, M.J. and M.J. Quinn. *Analytical Performance Prediction on Multicomputers*. in *Supercomputing'93*. 1993.
[Crovella99] Crovella, M.E., *Performance Prediction and Tuning of Parallel Programs*, in *Department of Computer Science*. 1999, University of Rochester.
[CSA98] Cardwell, N., S. Savage, and T. Anderson, *Modeling the Performance of Short TCP Connections*. 1998, Computer Science Department, Washington University.
[CW95] R.A. Coyne and R.W. Watson. The Parallel I/O Architecture of the High-Performance Storage System (HPSS). In *IEEE MSS Symposiu*m. IEEE Computer Society Press, 1995.
[DataGrid02] *The Data Grid Project, http://www.eu-datagrid.org, 2002.*
[DO00] Dinda, P. and D. O'Hallaron, *Host Load Prediction Using Linear Models.* Cluster Computing, 2000. **3**(4).
[Downey97] Downey, A. *Queue Times on Space-Sharing Parallel Computers*. in *11th International Parallel Processing Symposium*. 1997.
[Edwards84] Edwards, A.L., *An Introduction to Linear Regression and Correlation*. 1984: W.H. Freeman and Company.
[FK98] Foster, I. and C. Kesselman. *The Globus Project: A Status Report*. in *IPPS/SPDP '98 Heterogeneous Computing Workshop*. 1998.
[FSW+99] Faerman, M., A. Su, R. Wolski, and F. Berman. *Adaptive Performance Prediction for Distributed Data-Intensive Applications*. in *ACM/IEEE SC99 Conference on High Performance Networking and Computing*. 1999. Portland, Oregon.
[Globus02] *The Globus Project, http://www.globus.org, 2002.*
[Gkantsidis02] Gkantsidis, C. Parallel Download, http://www.cc.gatech.edu/~gantsich/parallel_download.htm, 2002.
[GM01] Guo, L. and I. Matta, *The War between Mice and Elephants*. 2001, Computer Science Department, Boston University.
[GP94] Groschwitz, N. and G. Polyzos. *A Time Series Model of Long-Term Traffic on the NSFnet Backbone*. in *IEEE Conference on Communications (ICC'94)*. 1994.
[GriPhyN02] *The GriPhyN Project, http://www.griphyn.org, 2002.*
[GS00] Gray, J. and P. Shenoy. *Rules of Thumb in Data Engineering*. in *International Conference on Data Engineering ICDE2000*. 2000. San Diego: IEEE Press.
[GT99] Geisler, J. and V. Taylor. *Performance Coupling: Case Studies for Measuring the Interactions of Kernels in Modern Applications*. in *SPEC Workshop on Performance Evaluation with Realistic Applications*. 1999.
[HD96] Harchol-balter, M. and A. Downey. *Exploiting Process Lifetime Distributions for Dynamic Load Balancing*. in *1996 Sigmetrics Conference on Measurement and Modeling of Computer Systems*. 1996.
[HJS+00] Hoschek, W., J. Jaen-Martinez, A. Samar, and H. Stockinger. *Data Management in an International Grid Project*. in *2000 Internationsl Workshop on Grid Computing (GRID 2000)*. 2000. Bangalore, India.
[Holtman00] Holtman, K. *Object Level Replication for Physics*. in *4th Annual Globus Retreat*. 2000. Pittsburgh.
[HP91] Haddad, R. and T. Parsons, *Digital Signal Processing: Theory, Applications and Hardware*. 1991: Computer Science Press.

[HS97] T.A. Howes and M.C. Smith. *LDAP Programming Directory Enabled Application with Lightweight Directory Access Protoco*l. Technology Series. MacMillan, 1997.

[HSS00] Hafeez, M., A. Samar, and H. Stockinger. *Prototype for Distributed Data Production in CMS*. in *7th International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT2000)*. 2000.

[JCD+00] Johnson, K., et al. The Measured Performance of Content Distribution Networks. in Proceedings of the 5th International Web Caching and Content Delivery Workshop. 2000. Lisbon, Portugal.

[Jones02] Jones, R. *The Public Netperf Homepage, http://www.netperf.org/netperf/NetperfPage.html*. 2002.

[KRR00] Kangasharju, J., K. Ross, and J.W. Roberts. *Performance Evaluation of Redirection Schemes in Content Distribution Networks*. in *Proceedings of 4th Web Caching Workshop*. 1999. San Diego.

[LLM88] M. Litzkow, M. Livny, and M. Mutka. Condor – A Hunter of Idle Workstations. In *Proc. 8th Intl Conf. on Distributed Computing System*s, pages 104–111, 1988.

[LSZ+02] Lamehamedi, H., B. Szymanski, S. Zujun, and E. Deelman, *Data Replication Strategies in Grid Environments*, *in 5th International Conference on Algorithms and Architecture for Parallel Processing, ICA3PP'2002*, Bejing, China, October 2002, IEEE Computer Science Press, Los Alamitos, CA, 2002, pp. 378-383

[LSZ+03] Lamehamedi, H., B. Szymanski, S. Zujun, and E. Deelman, *Simulation of Dynamic Replication Strategies in Data Grids*, in *12th Heterogeneous Computing Workshop (HCW2003)*, Nice, France, April 2003.

[LIGO02] *The LIGO Experiment, http://www.ligo.caltech.edu/, 2002*.

[Mach97] The Mach Project Home Page, http://www-2.cs.cmu.edu/afs/cs/project/mach/public/www/mach.html, 1997.

[Maxemchuk75] Maxemchuk, N.F. *Dispersity Routing*. in *Proceedings of the International Conference on Communications*. 1975.

[Merkey94] Merkey, P., *Beowulf Project at CESDIS*, http://beowulf.gsfc.nasa.gov/, 1994.

[ML90] Mak, V.W. and S.F. Lundstrom, *Predicting the Performance of Parallel Computations.* IEEE Transactions on Parallel and Distributed Systems, 1990: p. 106-113.

[MLB95] Malpani, R., J. Lorch, and D. Berge. *Making World Wide Web Caching Servers Cooperate*. in *Proceedings of the Fourth International WWW Conference*. 1995.

[MMR+01] Malon, D., E. May, S. Resconi, J. Shank, A. Vaniachine, T. Wenaus, and S. Youssef. *Grid-enabled Data Access in the ATLAS Athena Framework*. in *Computing and High Energy Physics 2001 (CHEP'01) Conference*. 2001.

[NetLogger02] *NetLogger: A Methodology for Monitoring and Analysis of Distributed Systems*. 2002.

[NM02] Newman, H. and R. Mount. *The Particle Physics Data Grid, www.cacr.caltech.edu/ppdg*.

[OM88] Ostle, B. and L.C. Malone, *Statistics in Research*. 1988: Iowa State University Press.

[PAD+02] Planck, J.S., et al., *Algorithms for High Performance, Wide-Area, Distributed File Downloads*. 2002, University of Tennessee, Department of Computer Science.

[Pankratz91] Pankratz, A., *Forecasting with Dynamic Regression Models*. 1991: John Wiley & Sons Inc.

[Rabin89] Rabin, M.O., *Efficient Dispersal of Information for Security.* Journal of the ACM, 1989. **38**: p. 335-348.

[RF01] Rangahathan, K., and I. Foster, *Design and Evaluation of Replication Strategies for a High Performance Data Grid*, in *Computing and High Energy and Nuclear Physics 2001 (CHEP'01) Conference*. 2001.

[Rizzo97] Rizzo, L., *Effective Erasure Codes for Reliable Computing.* Computer Communications Review, 1997.

[RKB00] Rodriguez, P., A. Kirpal, and W.E. Biersack. *Parallel-access for Mirror Sites in the Internet*. in *Proceedings of IEEE INFOCOM*. 2000.

[RLS98] R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed Resource Management for High Throughput Computing. In *Proc. 7th IEEE Symp. on High Performance Distributed Computin*g. IEEE Computer Society Press, 1998.

[Sandvine02] *Peer-to-Peer File Sharing: The Effects of File Sharing on a Service Provider's Network*. 2002, Sandvine Whitepaper.

[SB98] Schopf, J.M. and F. Berman. *Performance Predictions in Production Environments*. in *IPPS/SPDP'98*. 1998.

[SC00] Shen, X. and A. Choudhary. *A Multi-Storage Resource Architecture and I/O, Performance Prediction for Scientific Computing*. in *9th IEEE Symposium on High Performance Distributed Computing*. 2000: IEEE Press.

[Schopf97] Schopf, J.M. *Structural Prediction Models for High Performance Distributed Applications*. in *Cluster Computing (CCC'97)*. 1997.

[SDSS03] *Sloan Digital Sky Survey*, http://www.sdss.org/, 2003.

[SFT98] Smith, W., I. Foster, and V. Taylor. *Predicting Application Run Times Using Historical Information*. in *IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing*. 1998.

[SGG02] Saroiu, S., P.K. Gummadi, and S. Gribble. *A Measurement Study of Peer-to-Peer File Sharing Systems*. in *Proceedings of Multimedia Computing and Networking (MMCN'02)*. 2002.
[Speedera02] Speedera, http://www.speedera.com, 2002.
[SW02] Swany, M. and R. Wolski. *Multivariate Resource Performance Forecasting in the Network Weather Service*. in *Submitted for Publication*. 2002.
[SYSSTAT02] *SYSSTAT Utilities Homepage, http://perso.wanadoo.fr/sebastien.godard/, 2002.*
[Tanenbaum95] Tanenbaum. A., *Distributed Operating Systems*, Prentice Hall, Englewood Cliffs, NJ, 1995.
[TB86] Thomasian, A. and P.F. Bay, *Queuing Network Models for Parallel Processing of Task Systems.* IEEE Transactions on Computers, 1986. **35**(12).
[TBS+01] Thain. D, J. Basney, S. Son, and M. Livny. *The Kangaroo Approach to Data Movement on the Grid.* in *Tenth IEEE Symposium on High Performance Distributed Computing (HPDC10)*. San Francisco, California. August 2001.
[TF01] Tirumala, A. and J. Ferguson. *Iperf 1.2 - The TCP/UDP Bandwidth Measurement Tool, http://dast.nlanr.net/Projects/Iperf*. 2001.
[TGL+01] Tierney, B., D. Gunter, J. Lee, and M. Stoufer, *Enabling Network-Aware Applications.* in *10th IEEE Symposium on High Performance Distributed Computing (HPDC-10).* August 2001.
[Vazhkudai03] Vazhkudai, S., *Enabling the Co-Allocation of Grid Data Transfers.* in *4th International Workshop on Grid Computing (GRID 2003),* Phoenix, Arizona, November 2003.
[VS03] Vazhkudai, S., and J. Schopf. *Using Regression Techniques to Predict Large Data Transfers*. Journal of High Performance Computing Applications - Special Issue on Grid Computing: Infrastructure and Applications, 2003.
[VTF01] Vazhkudai, S., S. Tuecke, and I. Foster. *Replica Selection in the Globus Data Grid*. in *First IEEE/ACM International Conference on Cluster Computing and the Grid (CCGRID 2001)*. 2001. Brisbane, Australia: IEEE Press.
[Wang99] Wang, J., *A Survey of Web Caching Schemes for the Internet.* ACM Computer Communication Review, 1999.
[Web100Project02] *The Web100 Project, http://www.web100.org, 2002.*
[WPP02] Wang, L., V. Pai, and L. Peterson. *The Effectiveness of Request Redirection*. in *Proceedings of the 5th OSDI Symposium*. 2002.
[Wolski98] Wolski, R., *Dynamically Forecasting Network Performance Using the Network Weather Service.* Cluster Computing, 1998.
[YM01] Yilmaz, S. and I. Matta, *On Class-based Isolation of UDP, Short-lived and Long-lived TCP Flows*. 2001, Computer Science Department, Boston University.
[ZLP96] Zaki, M.J., W. Li, and S. Parthasarathy. *Customized Dynaimic Lad Balancing for Network of Workstations*. in *High Performance Distributed Computing (HPDC'96)*. 1996.
[ZMF98] Zhang, L., S. Michel, and S. Floyd. *Adaptive Web Caching: Towards a New Global Caching Architecture*. in *Proceedings of the Third International Caching Workshop*. 1998.
[ZQK00] Zhang, Y., L. Qiu, and S. Keshav. *Speeding Up Short Data Transfers: Theory, Architecture Support and Simulation Results*. in *NOSSDAV 2000*. 2000. Chapel Hill, NC.
[ZQK99] Zhang, Y., L. Qiu, and S. Keshav, *Optimizing {TCP} Start-up Performance*. 1999, Department of Computer Science, Cornell University.